National Aeronautics and Space Administration

System on a Chip Devices—FY10

Dr. Steven M. Guertin
Jet Propulsion Laboratory
Pasadena, California

Jet Propulsion Laboratory
California Institute of Technology
Pasadena, California

JPL Publication 10-20 12/10

National Aeronautics and Space Administration

System on a Chip Devices—FY10

NASA Electronic Parts and Packaging (NEPP) Program
Office of Safety and Mission Assurance

Dr. Steve M. Guertin
Jet Propulsion Laboratory
Pasadena, California

NASA WBS: 724297.40.49
JPL Project Number: 103982
Task Number: 03.04.09

Jet Propulsion Laboratory
4800 Oak Grove Drive
Pasadena, CA 91109

http://nepp.nasa.gov

This research was carried out at the Jet Propulsion Laboratory, California Institute of Technology, and was sponsored by the National Aeronautics and Space Administration Electronic Parts and Packaging (NEPP) Program.

Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise, does not constitute or imply its endorsement by the United States Government or the Jet Propulsion Laboratory, California Institute of Technology.

Copyright 2010. California Institute of Technology. Government sponsorship acknowledged.

TABLE OF CONTENTS

ABSTRACT

This is the final report for the fiscal year (FY) 2010 System-on-a-Chip (SOC) Devices NEPP task. This report describes testing of the Aeroflex UT699 LEON 3FT Sparc™ microprocessor for single event effects (SEE), including examination of anomalies and potential flux and fluence dependence. It describes the test system and reports on data collected for the register file, data cache, instruction cache, and SpaceWire single-event upsets. This task also included review and participation in the NASA test development for the Maestro 49-core multicore microprocessor produced by Boeing. Review of NASA test plans and recommendations for development efforts are made.

This task seeks to provide the NASA community with improved methods for SOC testing, with the goal of providing data that is of interest to NASA and that meets the standards of the wider radiation effects community.

NOTATION

This document uses several specific notations briefly described here.

- 0xDDDD_DDDD or 0xDD is used to express numbers in hexadecimal format. The digits 0–9 and a–f refer to values from 0 to 15. '_' is used to aid the eye.
- 0bDDDD_DDDD or 0bDD is used to express numbers in binary format. '_' is used to aid the eye.
- %l, %o, %g, %i are prefixes for registers in the Sparc™ architecture.
- Endianness is always little endian, indicating that digit 0 is the least significant digit and is at the trailing (right) end of the number, i.e., 0×1234 has 4 as digit 0 and 1 as digit 3.

1.0 BACKGROUND OF THE SOC TASK

This System-on-a-Chip (SOC) Devices NEPP task was started in FY2010 with the primary goals of (1) establishing appropriate test methods for SOC devices and related microprocessors for NASA, (2) investigating the appropriateness and methods for engaging device manufacturers in the testing of their SOCs, (3) engaging the NASA community to determine the most desired test data to be collected from SOC and microprocessor devices, and (4) surveying available SOC devices, primarily microprocessor-based SOCs, to validate test methods and identify general strengths or weaknesses in various architectures.

2.0 SOC TESTING INTRODUCTION

This report concentrates on testing of SOC devices for single-event effects (SEEs). Understanding and measuring SEE cross sections is important in both commercial and radiation-hardened-by-design (RHBD) SOCs for slightly different reasons. Both general types of devices share test complexity issues where both types may be considered together. However, there are also differences in device structure and operating situations that force different test methods involved in measuring SEE.

Due to the complex nature of SOC devices, full test suites can be enormous [1]. This necessitates picking a general structure for testing that allows characterization of the most important upset types. For both commercial and RHBD devices, the most important upset types are easy to state but hard to identify and measure. The most important upset types include the types that will manifest most often in a system.

The first approach in test development for SOCs is to measure the upset rates for the parts that will be used the most. This means identifying the primary capabilities at play in selecting the given SOC and specifically testing operation of those capabilities. For microprocessor SOCs, this means testing the microprocessor in a situation where it can exercise the on-chip memory management unit to access external memory, and where it can communicate through an interface of interest to NASA.

At this point, commercial and RHBD SOC test structures must deviate. Commercial devices are not expected to handle upsets in a recoverable way, so the system required for testing such a device targets measuring the weakest elements in a bottom-up way. The cross sections for various static storage elements are measured. Then, dynamic operation is tested cursorily to verify that the static rates will dominate the upset rates in a real system. It is expected that static rates will dominate, especially because commercial devices are almost always used with derated operating speed. In the unlikely event that dynamic upsets dominate the event rates for a commercial device, test systems must then target these dynamic upset modes using codes that mimic real operation. (This will be expanded in the future efforts of this task, provided a suitable test vehicle can be found.)

For RHBD devices, it is common but not necessarily beneficial to base the test system on static elements. This approach is inappropriate because if the RHBD techniques work, the system should be immune to static upsets. It is also difficult to test RHBD systems for dynamic upsets because the system may be RHBD under a certain target SEE rate, but testing will use elevated upset rates, leading to issues where rare detection of device errors may be due to overwhelming of fault-tolerant (FT) features of the RHBD systems.

This report targets the testing of RHBD systems (utilizing the Aeroflex UT699 and Boeing Maestro devices). The priorities for testing of these devices are listed below, in order.

1. Measure the static upset sensitivity for all static elements (for the most desired functional blocks).
2. Verify the operation of the FT elements of the system (verify upsets to static elements are corrected by fault handling processes).
3. Test the device with functional test codes utilizing desired elements of the system (including peripherals). These tests must include correct operation of the RHBD/FT elements.

4. In the event unexpected errors occur in step 3, identify if the errors could be caused by overwhelming the RHBD/FT elements. Repeat testing with multiple fluxes and fluences to eliminate dependencies.

It is very important to point out that uncorrected errors in RHBD/FT systems may be very difficult to evaluate in terms of flux or fluence dependence. This is due to one main problem: a low cross section. If an error type has a very low cross section, it will be difficult to reduce flux or fluence and still see the event. Even if the cross section is not that small, it may still be difficult to evaluate flux or fluence dependence if the device has a much higher cross section for upsets that are fixed by the RHBD elements. In this case, if the device is seeing hundreds or thousands of upsets in protected elements, it is not a long stretch to say that an uncontained error is due to multiple upsets overwhelming the FT elements. In either case, it will be difficult to obtain significant numbers of events to draw conclusions.

Two avenues exist in exploring an unfixed error. The first is to simply alter the flux and/or fluence and observe if the cross section remains the same. The second is to identify the signature of the error and determine how it could have been caused by overwhelming of the RHBD. The former requires 10–20 upsets in each set of test conditions while the latter only requires a few well-documented events. This study used both methods to evaluate the unfixed errors in the UT699.

3.0 DEVICES FOR STUDY

For FY2010, work was split between the Aeroflex UT699 LEON 3FT Sparc™ V8 SOC, and Boeing's Maestro 49-core Tilera-based multicore microprocessor. The UT699 was originally tested by Aeroflex with JPL participation in 2008. During the 2008 testing, some anomalous events were observed, which were postulated to be due to multi-bit upsets due to high flux. To address the postulate and expand the test results of the on-chip peripherals, the UT699 was tested this year in March and August for SEEs. The Maestro device is scheduled for radiation testing in FY2011. The work in FY2010 consisted of participation in test planning and development of the Boeing and NASA SEE test efforts for FY2011.

3.1 UT699

The UT699 is Aeroflex's implementation of the Gaisler LEON 3 Fault Tolerant core, mated to several peripheral components. The UT699 is a SOC device with a built-in memory management unit and several peripheral devices. Figure 3.1-1 shows the block diagram of the UT699.

The reason for including the UT699 is that, as an RHBD SOC that provides immediate opportunities to work with the manufacturer, it is an excellent case study for examining the benefits involved in manufacturer participation. In this case, testing of the UT699 was carried out in 2008 as part of an initial effort to characterize the radiation response of the device. Events that could not be clearly identified as being due to single or multiple ions were collected. The dataset contained significant questions that were hoped to be resolved by testing the device under the NEPP SOC devices task.

Testing of the UT699 could include a lot of specialized software to target the various peripherals. However, the current effort leveraged previous testing that indicated some anomalous behavior at low linear energy transfer (LET). Anomalies were known to be present in testing that targeted only the microprocessor; therefore, this was the initial goal of the test development for the UT699. As a result, it was found that the anomalies were not just test artifacts, and the selection of the microprocessor as the main test target was justified.

In addition to the microprocessor, testing also targeted the SpaceWire ports in order to provide some hardware specific test results.

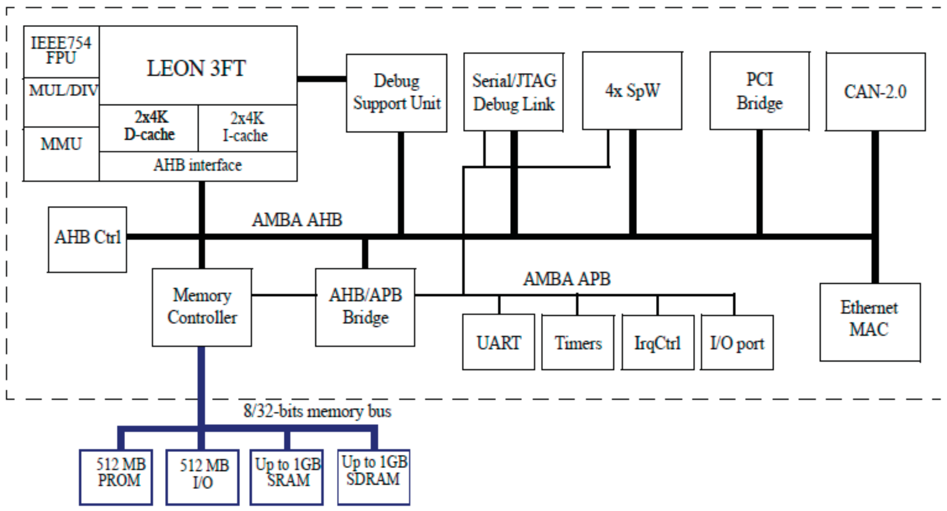


Figure 3.1-1. Block diagram of the UT699.

3.2 Maestro

The Maestro device is Boeing's 49-core Tilera-based microprocessor that is being built on Boeing's 90 nm RHBD library. In addition to the 49 cores, Maestro also has four memory managers, four XAUI ports, and a host of other peripherals. The structure of the Tilera Tile-64, as shown in Figure 3.2-1, is very similar to the Maestro except for the replacement of PCIe with more XAUI.

Maestro has the potential to be a very important device for space programs and may be used in NASA missions in the near future. Maestro provides very high computational capability (20GOPs) and is a potential enabling architecture that is of significant interest to NASA at this time. Because of this, and Maestro's SOC structure, it is a natural fit for this task. During FY10, JPL supported review meetings for Maestro and provided input on SEE test software architecture, including recommendations on overall architecture and migration of specific codes from the Board Test Kit (BTK) from Tilera, which supports functional verification of Tilera-based microprocessor devices.

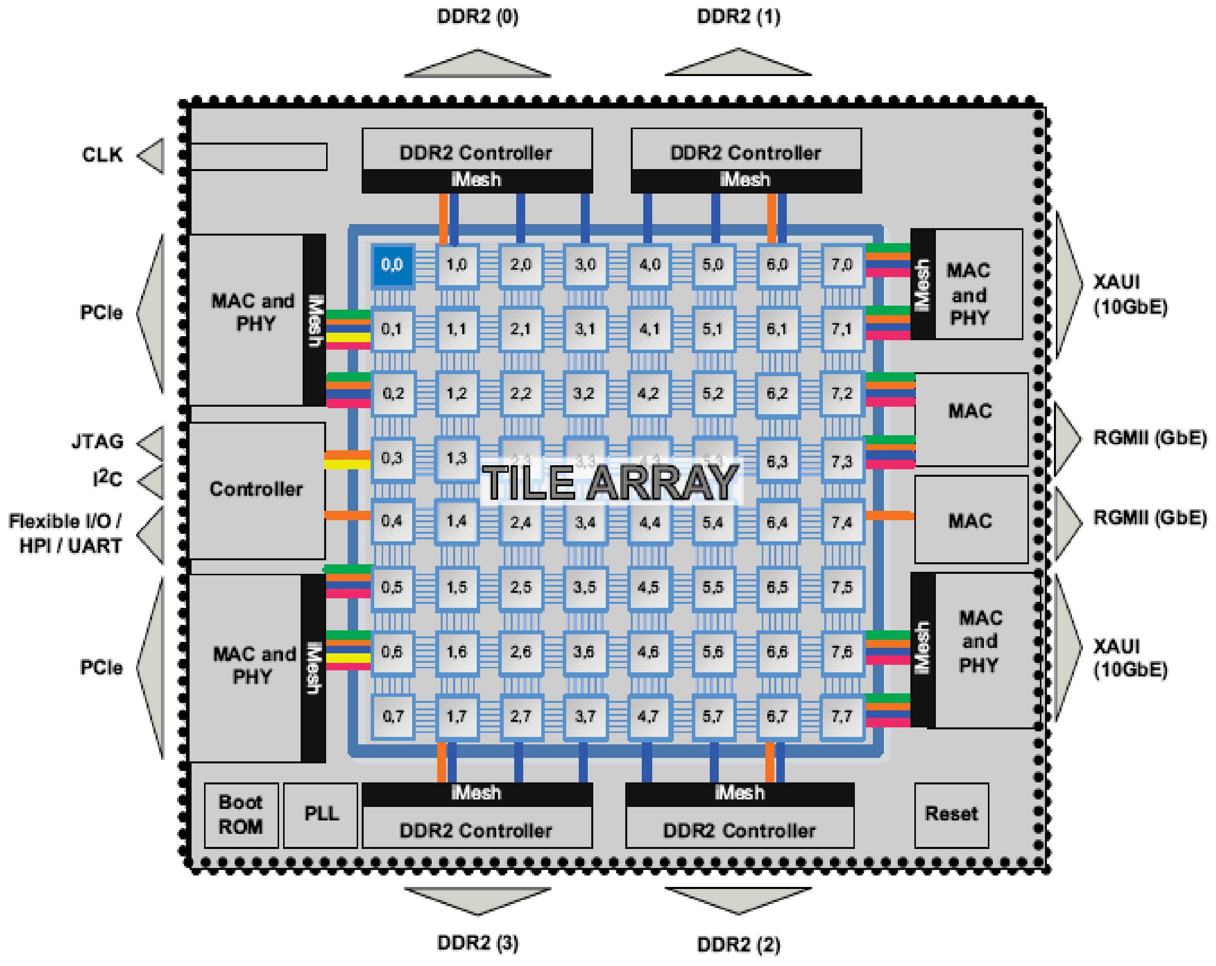


Figure 3.2-1. Tilera Tile-64 layout.

4.0 TEST SETUP

The FY2010 SOC devices task tested the UT699 for SEE. This section discusses the test setup.

4.1 Physical Setup

The UT699 was tested using the GR-CPCI-UT699 test board. Only one device under test (DUT), mounted on the board, was used for testing. The test board was modified by changing jumpers to allow separate power supply to the core of the UT699 in order to monitor for high current conditions.

The board was powered with the core voltage set at 2.3 V and the input/output (I/O) set at 3.0 V, the specification minimum for operation. Some of the runs were performed with these voltages at 2.5 V and 3.3 V, leading to a reduced cross section at the threshold region.

The UT699 was tested at the Texas A&M University Cyclotron (TAMU). Testing was performed at normal incidence and 60 degrees, using a range of ions. Table 4.1-1 provides the list of ions.

Figure 4.1-1 shows the test setup for the UT699. The basic form for hardware setup is common to all test software versions. The hardware changes from test to test based on the embedded mini test performed (see Section 4.2.1.1).

Table 4.1-1. The ions used for testing of the UT699

Ion	Energy (MeV/nucleon)	Angle (degrees)	LET_{eff} ($MeV\text{-}cm^2/mg$)
Ar	15	0	8.7
Ar	15	48	13.2
Ar	15 (degraded to 5)	48	21
Kr	15	0	30
Kr	15	60	60

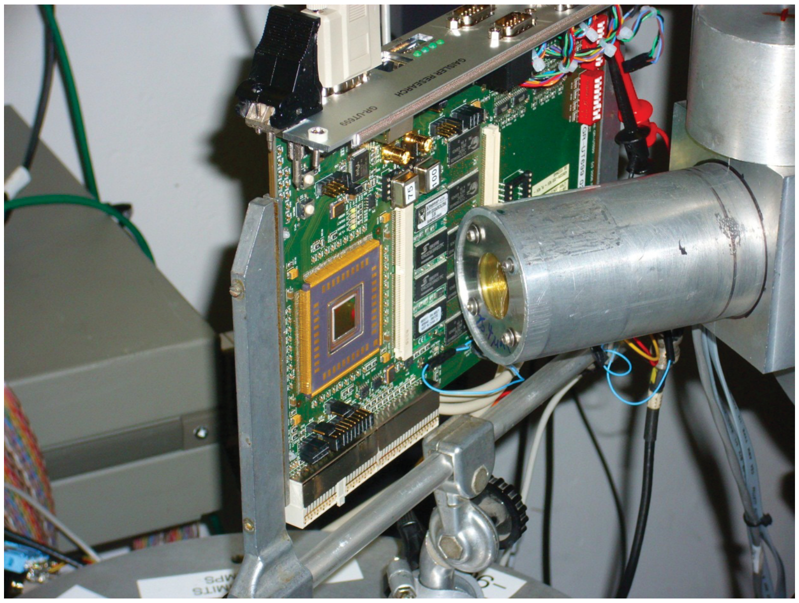


Figure 4.1-1. The configuration of the UT699 at 60 degrees at the end of the beam line at TAMU.

4.1.1 Considerations for Instruction Cache Test

The instruction cache mini test requires no special hardware considerations.

4.1.2 Considerations for SpaceWire Test

The SpaceWire test requires a simple cable connection between SpaceWire ports number 1 and 2.

4.2 Software Setup

The UT699 test system operating software breaks down into three pieces. The first piece is the test software operating on the GR-CPCI-UT699 test board. The second is the software program used to interface with the UT699 test software via the RS232 port. The third is the power supply system used to control the power delivery to the test board.

4.2.1 Test Software for Operation of UT699

The test software for operation of the UT699 consists of a basic architecture that is augmented with special test codes for targeting particular elements in the UT699. The basic architecture is named SGMAN.

4.2.1.1 Basic SGMAN Architecture

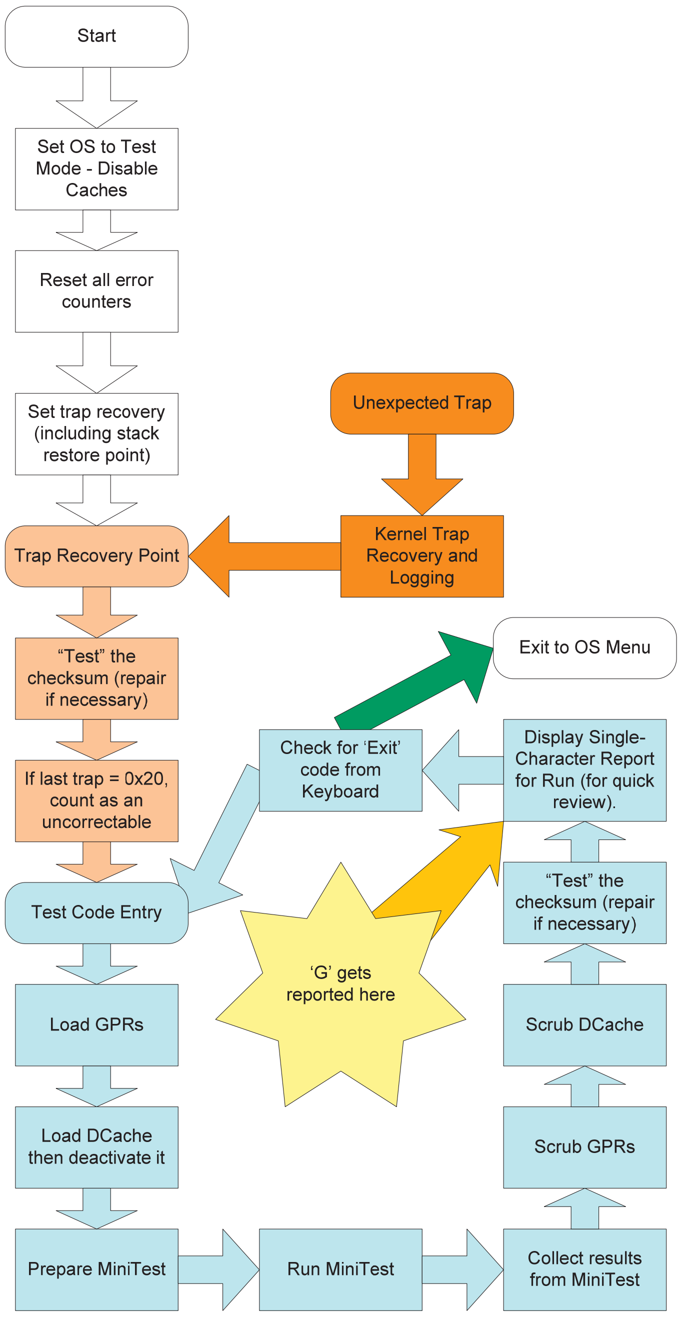
This section defines the SGMAN test flow. Before this can be defined, however, the context in which the SGMAN test flow runs must be understood. That is, the test flow runs on top of a micro-operating system. SGMAN follows a generic implementation of the test operating system. In this operating system, there is a micro-kernel (providing only basic stack operations), operating system state handling (i.e., if the operating system is in "test mode," then the trap handlers return to the test program), character I/O to the Universal Asynchronous Receiver/Transmitter (UART) (supported via direct operation, no interrupts), and system-level code checksum (which is different from test-level code checking).

The following "support" features are provided to test systems:

- Support for all trap handlers (a feature provided by SGMAN).
- Preparation of a "safe area" for code execution by implanting the "test code" in a sea of software traps that will cause a trap condition in the event of a bad branch or program counter (practically speaking, only about 50–75% of these can be captured because only some of the PC bits cause a clear jump out of the code without going to illegal memory space) (a feature provided by SGMAN).
- Auto-copying of test code to the "safe area," and auto-recovery of modified code in the event of traps (a feature provided by SGMAN).
- A test log that handles recording of important event information from test programs in a standard way (SGMAN provides the feature, but it cannot be used during register and cache scrubbing due to inconsistent operating state [i.e., register and cache scrubbing require some resources which the test log routines invalidate]).

The Architecture for Microprocessor Functional Radiation Testing (AFMRT) also specifies a "middle" support package that provides methods for displaying text blocks and log file. The middle support package also provides the menu system, which allows the user to set the test program configuration.

Given all of the support features described above, a test is begun when the middle support package is used to start a test. Figure 4.2-1 shows the test flow at that point.



Start
Set OS to Test Mode - Disable Caches
Reset all error counters
Set trap recovery (including stack restore point)
Unexpected Trap
Kernel Trap Recovery and Logging
Trap Recovery Point
Exit to OS Menu
"Test" the checksum (repair if necessary)
Check for 'Exit' code from Keyboard
Display Single-Character Report for Run (for quick review).
If last trap = 0x20, count as an uncorrectable
Test Code Entry
'G' gets reported here
"Test" the checksum (repair if necessary)
Load GPRs
Scrub DCache
Load DCache then deactivate it
Scrub GPRs
Prepare MiniTest
Run MiniTest
Collect results from MiniTest

Note on MiniTest:

For basic SGMAN, the MiniTest is to prepare a special location for cached instructions, then run in that location with ICache active for a set number of passes before completing the run of the MiniTest

For HERT_SW, the MiniTest is to reset the spacewire ports, negotiate connection at full duplex, prepare data images for transfer (1 source region for port 1, and 1 for port 2, clear out the target regions), tell SpaceWire to send the data full duplex, gather the metrics from the ports, and verify the contents of the target regions match the source regions.

Figure 4.2-1. The architecture of the SGMAN test system running an arbitrary mini-test. (The indication of where "G" gets reported shows where the system indicates that a register is modified by 16 bit upset—see Section 7.)

4.2.1.2 Test System Operation

The general structure of SGMAN allows targeted handling of upsets in different ways. This subsection describes how the following upset modes are handled for the UT699 test system, V 0.2.x.

System Software Corruption

The actual software image is protected during execution by limiting access to it. All calls to system software during testing will be made with the instruction cache disabled. This results in a small probability of upsets leading to corruption. In order to quantify these, the software image will be checksummed on every pass, and any change in the checksum will be reported (in development: Priority 3). An additional copy of the software will be made before the test begins, and both versions of the software will be checksummed during each pass. Any time a copy's checksum is incorrect, that code is recopied to fix the error. When they both disagree, it will be assumed the golden checksum has been corrupted and will be rewritten.

Register Upsets

Register upsets should be transparent to the operating software, as long as the registers are "scrubbed" periodically. For this test system, the general purpose registers (GPRs) are loaded before the main test loop. After that, the GPRs are "scrubbed" during the end of a pass code. Scrubbing is selectable between copying registers to themselves (using the "mov" instruction), and storing the registers to memory. At the end of scrubbing, the "correct" value is stored to all the GPRs. Although all registers are scrubbed during operation, the error detection and correction (EDAC) counter is reset before "scrubbing" at the end of each pass. During the pass-end scrubbing, only six of the windows (6*16) registers are actually scrubbed (96 total, 3744 bits including EDAC). The error records are reported PER WINDOW. Thus, for a 0-error pass, six entries with zero errors will be stored.

Traps

The test system is configured so that all trap types except for software traps are captured and counted (and logged). The caches are configured so that they are frozen when the trap occurs and will not run out of the caches. During the trap handler, the caches are disabled and the trap handler returns to the start of pass code (essentially starting a new pass). 0x100 trap handlers are initialized to operate in this way.

Data Cache Upsets

Bulk data cache upsets are spotted by the processor due to parity errors that cause cache misses. Another type of data cache upset, the tag upset, in which the wrong line appears to be in the cache, will look identical to the first type of upset (if a parity upset is spotted, it will be counted as a cache tag parity event instead of a cache data parity event). To catch all types of data cache upsets, the cache is tested by loading a pattern, deactivating the cache, soaking, then activating and reading the cache. The structure of the data cache test is as follows: The test system is started and runs up to the point where it is in a pass. Once in the pass, both caches are flushed using a flush command. Then, the data cache is loaded, and the area under the cache is loaded. The pass is then allowed to continue and do its other work. When the soak portion of the pass and the lower level tests/data collection are complete, the data cache is interrogated for upsets. Upsets are recorded during reading of the cache using the built-in error counters, but the data are also checked for upsets and for matches to background data. It is expected that upsets will result in data matching the background pattern. The data cache is 8 kB, with 16 b/line. The cache is not automatically disabled when a trap occurs, so the test software by default turns it off in the trap handler. (Traps are supposed to freeze the caches, but the cache condition register [CCR] is not changed upon entry into a trap handler, and it is not clear that they are really frozen. In fact, with the asynchronous freeze setting enabled, the test verified that illegal instructions still cause the trap handler to be loaded into the instruction cache.)

Instruction Cache Upsets

The instruction cache test is the most nested/inner part of the test system. In fact, due to its nature, it is considered the "test"—that is, the instruction cache test is the target test code of the test software that has the code. The instruction cache is tested by running software in a controlled way through the instruction cache while the rest of the test system is running. The instruction cache test works by

1. Having the instruction cache flushed before a test pass
2. Loading the "test safe area" with trap 0x82's—these will cause a pass restart if test code region is breeched
3. Writing a noop loop into the "test safe area," which scrubs the parity counters and ends by retl (the caches are turned off following the retl)
4. The noop loop is nearly the size of the entire cache, and code execution should not leave the cache
5. When any trap is hit, the instruction cache is turned off in the trap handler (the processor does not lock down the caches for many traps (e.g., an illegal instruction will not freeze the cache)
6. After a given number of loops on the code in the instruction cache, the state of the instruction cache error counters is stored as a frequency distribution
7. At the end of a test pass, the upset stripchart is interrogated and the upset events recorded

Configuration Registers

This part is not yet written. A list of all necessary configuration registers and masks for their important bits is used to create an image of the configuration before the run goes into the pass phase. In the pass phase, all registers are read and their important bits scrubbed for errors. Any errors are logged. In addition, any important bits will be reset to their start-of-run values. As of right now, the scrubbed registers are psr (all bits), PCR (0x6000), RPCR (0x3fff), wim (0xff bits), TBR (0xfffff000), M/DIV (not scrubbed), FSR (0x3fcfefff), CCR (0x1ff9ffff), (ICCR & DCCR not scrubbed), UARTCTRL (0x6bf Bits), UARTSPEED (0xfff Bits), MCFG1 (0x7efbcbff), MCFG2 (0xfffffeff), MCFG3 (0xc7ffffff).

4.2.1.3 Instruction Cache Test

The primary test code used for testing the UT699 is the instruction cache test. This can be seen in the detail area of Figure 4.2-1. The instruction cache test prepares a loop that executes in the instruction cache. This loop scrubs the instruction cache, and reads out the event counter registers so that the instruction cache upset counter does not overflow. The act of cycling through code in the instruction cache guarantees that the information in the cache is parity tested often, to enable automatic identification of cache upsets.

In the case of dormant code stored in the cache but not cycled through, it would be possible for the code to amass multiple bit upsets in the same parity protected element, leading to corrupt instructions that could not be captured by the fault tolerance system. Due to the event rates in terrestrial testing, such a corruption is expected in dormant code. Thus, the test was designed to have no dormant code.

4.2.1.4 HERT_SW Test

Figure 4.2-1 also shows the mini test for operation of SpaceWire. This mini test resets the SpaceWire connections of its two ports, renegotiates the connection between the two, prepares data images for transfer, instructs the SpaceWire controller to perform the transfer, and when the transfer is complete, verifies the images transferred.

5.0 TEST RESULTS

SGMAN and HERT_SW were used to collect test results from the UT699 and bin them in terms of different types of upsets.

5.1 Register File Corruptions

SGMAN is designed to catch bit errors in the register file by three means. First, it prepares the register file with known data and interrogates the values looking for discrepancies. Second, it monitors the register file error counter and keeps it clear in order to avoid overflow problems (the internal counter only counts the first six errors then overflows). Third, SGMAN monitors the EDAC uncorrectable trap (0×20) and counts these events.

SGMAN tests the register file by writing 0xffff_ffff or 0x0000_0000 to six of the register file windows in the UT699. Each window is prepared with its local registers (%l) and output registers (%o) loaded with the desired test pattern. Upon loading with a given pattern, the register file also prepares the seven EDAC check bits. The check bit definition in Table 5.1-1 is given in [1] as the EDAC architecture used by the UT699.

Given this definition, it is important to note the check bits for the standard values encountered by the test software given in Table 5.1-2.

The remainder of this section presents results from the register file testing.

Table 5.1-1. The EDAC check bit architecture used by the UT699.

CB0 = **D0 ^ D4 ^ D6 ^ D7 ^ D8 ^ D9 ^ D11 ^ D14** ^ D17 ^ D18 ^ D19 ^ D21 ^ D26 ^ D28 ^ D29 ^ D31
CB1 = **D0 ^ D1 ^ D2 ^ D4 ^ D6 ^ D8 ^ D10 ^ D12** ^ D16 ^ D17 ^ D18 ^ D20 ^ D22 ^ D24 ^ D26 ^ D28
CB2# = **D0 ^ D3 ^ D4 ^ D7 ^ D9 ^ D10 ^ D13 ^ D15** ^ D16 ^ D19 ^ D20 ^ D23 ^ D25 ^ D26 ^ D29 ^ D31
CB3# = **D0 ^ D1 ^ D5 ^ D6 ^ D7 ^ D11 ^ D12 ^ D13** ^ D16 ^ D17 ^ D21 ^ D22 ^ D23 ^ D27 ^ D28 ^ D29
CB4 = **D2 ^ D3 ^ D4 ^ D5 ^ D6 ^ D7 ^ D14 ^ D15** ^ D18 ^ D19 ^ D20 ^ D21 ^ D22 ^ D23 ^ D30 ^ D31
CB5 = **D8 ^ D9 ^ D10 ^ D11 ^ D12 ^ D13 ^ D14 ^ D15** ^ D24 ^ D25 ^ D26 ^ D27 ^ D28 ^ D29 ^ D30 ^ D31
CB6 = **D0 ^ D1 ^ D2 ^ D3 ^ D4 ^ D5 ^ D6 ^ D7** ^ D24 ^ D25 ^ D26 ^ D27 ^ D28 ^ D29 ^ D30 ^ D31

Note: The check bits given are given explicitly for the off-chip memory EDAC bits. This task has inferred this structure in check bits for the register file. Bold text indicates the lower 16-bits.

Table 5.1-2. EDAC check bits for standard values encountered in SGMAN.

Value	Check Bits
0x0000_0000	0b000_1100
0xffff_ffff	0b000_1100
0x0000_ffff	0b000_1100
0xffff_0000	0b000_1100

Notes: It is not an error that the check bits are the same for all standard values.
It is also important to note that if the check bits are all set to 0, it is a double-bit error.

5.1.1 EDAC Corrections

EDAC corrections appeared during testing. Table 5.1-3 summarizes the results and provides the fluence applied to the UT699 for each of the tested LETs and the resulting number of EDAC corrections.

Table 5.1-3. Observed SBUs in the register file.

LET (MeV-cm²/mg)	Fluence (#/cm²)	SBUs	Cross Section (cm²)
8.7	7.5e7	0	<1.3e-8
13.2	2.0e8	1	5e-9
21	5.6e7	0	<1.8e-8
30	1.2e7	18	1.6e-6
60	9.0e5	53	5.9e-5

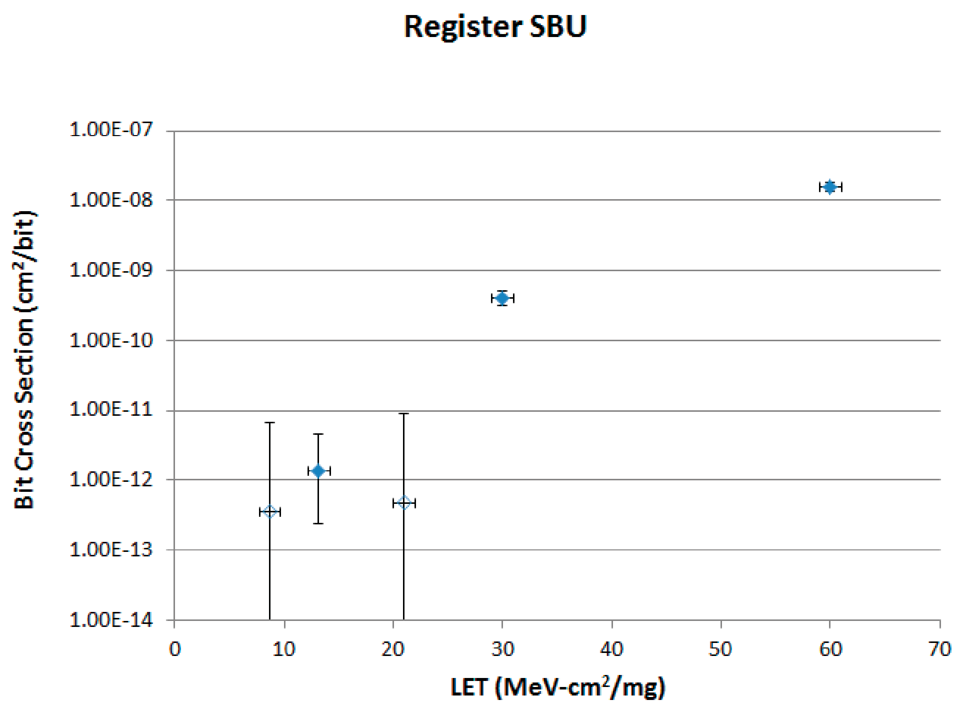


Figure 5.1-1. The sensitivity of register file bits is given. The data are consistent with test chip data for the storage elements that hold the register file data. The points at 8.7 and 21 are based on no counts.

It can be seen Table 5.1-3 that the threshold for events with EDAC correction is between LET = 21 and 30 MeV-cm^2/mg. This is consistent with the cross section for the storage elements employed by Aeroflex to store register file data.

The data from Table 5.1-3 are plotted in Figure 5.1-1.

5.1.2 EDAC Traps

Due to potential flux and program state issues, the data from trap 0×20 is considered largely inconclusive. In almost every case, these traps are consistent with the data corruption event discussed in the next section on uncontained errors.

5.1.3 Uncontained Errors

In the event that the test software detected a register with incorrect data but there was no associated trap 0×20, the observed corrupt register was examined for bit errors. SGMAN gives a count of the number of events where a register was incorrect, and a count of the number of bits that were wrong in the register file. SGMAN was written with the idea that these two counts could be compared and single- or double-bit errors would be obvious.

The only uncontained errors observed by SGMAN in the register file were 16-bit errors.

The uncontained errors are thus due to partial register reset events where one of the three 16-bit primitives that contain a register's value is set to all 0's. In runs where 1's are used as the pattern, two of the three primitives result in 16-bit register errors, while the 3rd primitive results in an EDAC trap if hit. When 0's are stored in the registers, only the EDAC trap is seen. For this reason, only the 1's data were used to generate the uncontained upset cross section discussed here. Figure 5.1-2 shows the cross section for these events.

It was hypothesized that the register partial reset event may be flux or fluence related. Data were taken to specifically address this question at LET = 13.2 MeV-cm^2/mg. For this test, the fluence of each run was reduced by a factor of 4 and the flux by a factor of 33. Both of these changes, if either dependence existed, would tend to reduce the cross section (by about 4 for the fluence change, and 33 for the flux change). Although the statistical significance is not strong, the resulting data point is not consistent with flux or fluence dependence and such dependencies are ruled out.

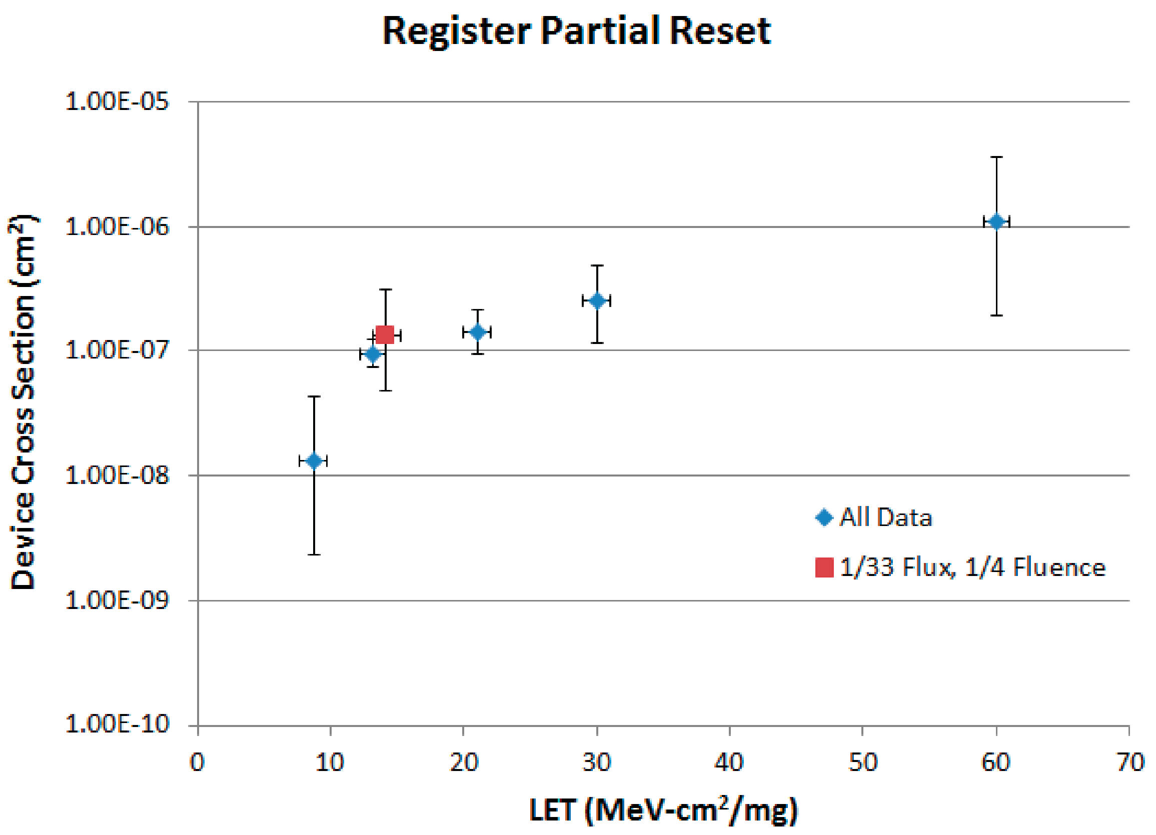


Figure 5.1-2. The cross section for register partial reset events is shown (for all runs where the test pattern was 0xffff_ffff). Also on this plot is a data point taken to test the hypothesis that the event was flux or fluence related (offset horizontally by 1).

5.2 Data Cache

The data cache was tested by storing data in it, deactivating it during irradiation, then reactivating and reading for upsets following exposure. Due to this structure of operation, the data cache was observed to be less sensitive, on a bit-for-bit basis, than the register file and instruction cache.

Table 5.2-1 shows the results for data cache testing. The results are also combined with the instruction cache and register results to generate the comparison plot in Section 5.4.

Table 5.2-1. Results for data cache testing of the UT699.

LET_{eff}	Fluence	Upsets	Cross Section
8.7 MeV-cm^2/mg	7.5e7	0	0
13.2	2.5e8	4	1.6e-8
21	5.1e7	7	1.4e-7
30	1.2e7	620	5.3e-5
60	9.0e5	904	1.0e-3

5.3 Instruction Cache

The instruction cache was tested using special test code particularly targeting it. Unlike the data cache, the instruction cache test code keeps the instruction cache active all the time. The reason for this is that in order to test the instruction cache, the processor must execute code from it. Thus, it is not reasonable to allow the instruction cache to integrate errors during a run the way the data cache does. If it were to integrate that way, the likelihood of hitting a corrupt instruction that was not detected would be very high and would lead to program crashes.

Table 5.3-1 shows the results for the instruction cache only.

Table 5.3-1. Results for instruction cache testing of the UT699.

LET_{eff}	Fluence	Upsets	Cross Section
8.7 MeV-cm^2/mg	7.5e7	1	1.3e-8
13.2	2.0e8	35	1.8e-7
21	2.7e7	37	1.4e-6
30	5.6e6	456	8.1e-5
60	4.5e5	380	8.4e-4

5.4 Combined SRAM Cell Results

The instruction cache information and the information from the data cache and register file are combined to generate the cross section for upsets in the protected elements of the UT699 and compared to previous results from [2] in Figure 5.4-1. The results from the instruction and data caches are expected to be the same; however, the data indicate the instruction cache is more sensitive than the data cache. This is likely due to differences in test operation of the caches. The results here, when compared to [2], indicate that the static nature of the data cache testing in Section 5.2 does not represent worst-case conditions for the data cache bits. Instead, the data cache should be used for storing and retrieving data during test.

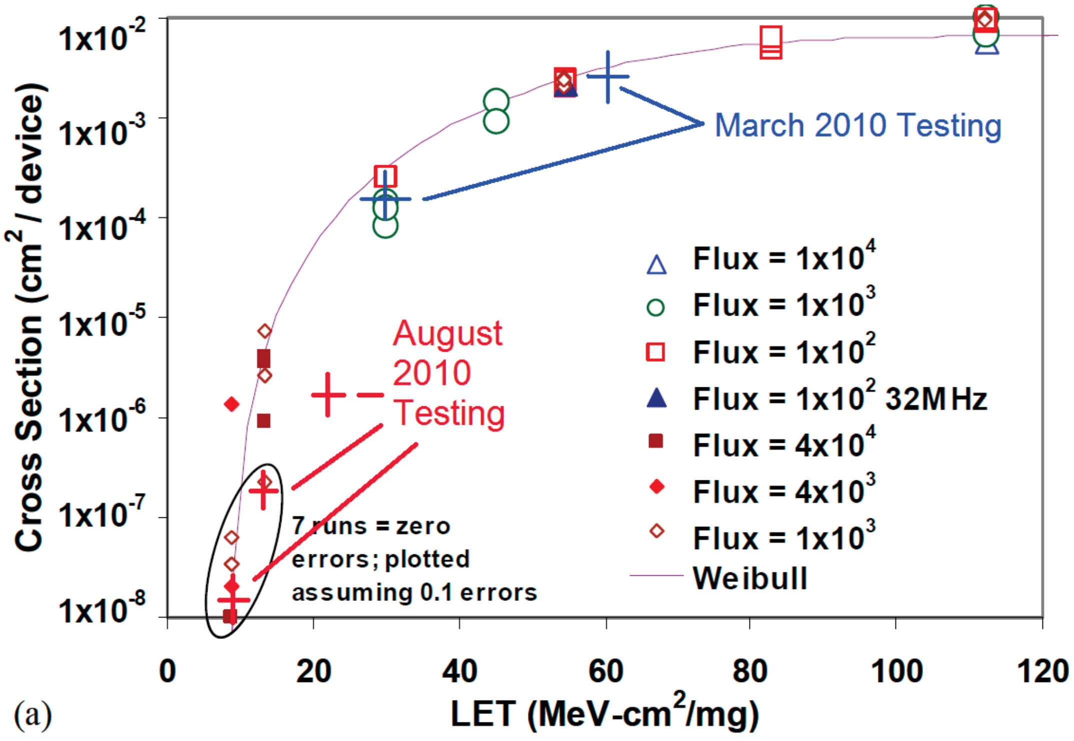


Figure 5.4-1. The cross section for all protected elements of the UT699 is shown compared to [2]. The only significant discrepancy is the LET = 21 point, which has no direct comparison with earlier data.

5.5 Watchdog Timer

The UT699 has a watchdog timer circuit that can be configured to connect its output to the UT699's reset circuit. Based on discussion with Gaisler Research, it was determined that a test of this reset mechanism would be of interest to potential users, since the device will likely fly with the watchdog timer used to monitor health of the UT699. The test was conducted using the SEU32 software developed by Gaisler Research. (For information about SEU32, contact Aeroflex.)

Testing discovered the UT699 watchdog timer does not catch all lockup conditions in the device. Testing was conducted at extremely elevated rates (2×10^4 ions/cm^2/sec) at LET = 31 MeV-cm^2/mg and also at 60. At 60, the results were inconclusive because event rates were so high, and event types so intrusive (the hardened flip flops upset at 60 leading to very unstable operation under high flux) that watchdog failures were very common and would mask an event where the processor stopped operating correctly but the watchdog continued to function. Thus, if an event locked up the processor in such a way that the watchdog was still being reset by the software, it would not be seen since another event could quickly occur, taking out the watchdog as well.

At LET = 30, it was observed that events could occur to compromise the operation of the system without reaching the threshold where the watchdog would realize the lack of operation. Two such events were seen. The cross section for the watchdog failing to fire in a compromised condition was about 5×10^{-7}cm^2. It is believed this is entirely due to the fact that a single watchdog circuit operating on the test chip is not sufficient for determining the status of the test system. This could be improved by increasing the portion of the UT699 that must be operating correctly in order for the watchdog verification code to fire.

It is unclear how important an on-device watchdog timer is for designers. Off-chip watchdog circuits are expected to be used in any potential NASA application. It is also true that the watchdog reset code could be required to check more health parameters of the UT699 before resetting the counter. Under the SEU32 code, the only requirement for watchdog reset is that the Leon 3FT core and memory management unit (MMU) are functioning.

5.6 SpaceWire

Two of the four ports of the UT699 were tested for radiation sensitivity using the HERT_SW test software. This software puts the SpaceWire ports into a full duplex data transfer operation and verifies correct data is transferred. The software is set to operate in an infinite loop of SpaceWire transfers, and then the beam is started.

The SpaceWire testing was performed on four of the test LETs. Table 5.6-1 summarizes the results.

Table 5.6-1. Summary of the results of the SpaceWire testing.

LET_{eff}	Fluence	Transferred Bits	Bit Errors
13.2	4.2e7	1.4e10	0
21	2.4e7	9.4e9	0
30	6e6	4.2e9	1
60	4.5e5	1.5e10	0

5.7 Other SEEs

Several other anomalous events were observed, as listed in Table 5.7-1. None of these events could be conclusively identified as a specific sensitivity separate from those already observed in the register file and cache when coupled with test flux and the possibility for test software bugs that are only observed in the case of rare upset types. (In this case, the special code for recording certain types of rare events may have led to some of the results listed.) These events are all consistent with results of partial reset of control or data registers used by SGMAN (this does not mean to imply a cause was found).

The information presented in Table 5.7-1 includes data across all runs where the event could be reasonably identified (except for the crashes which by definition are not related to a particular identifiable upset type). The fluences and LETs that apply to all entries in Table 5.7-1 are given in Table 5.7-2. It is important to note that the statistics involved are consistent with each event type being equivalently likely at any of the tested LETs.

Table 5.7-1. Other anomalies observed during SEE testing of the UT699. Counts presented are after best efforts to eliminate events that were probably due to software bugs. These data were collected across all runs at all LETs. See Table Table 5.7-2 for the associated fluences.

Anomaly	Occurrences & LETs
Crash (no other information)	1 @ 13.2, 1 @ 21, 2 @ 60
Trap 7 (memory address not aligned)	1 @ 13.2
UART Speed Register Corruption	1 @ 30
Program Counter Corruption	1 @ 30
Trap 0x12 (Level 2 interrupt)	1 @ 30

Table 5.7-2. Summary of the beams and fluences for the other SEE types who's event counts are given in Table 5.7-1. These LETs and fluences apply to each entry in Table 5.7-1.

LET_{eff}	Fluence
8.7	7.5e7
13.2	2.6e8
21	5.6e7
30	1.2e7
60	9.1e5

The events discussed here include primarily those where SGMAN was able to identify a state change in the UT699 contributing to the event. In the case of crashes, no further information is available and no such state change was recorded. It is not known if later updates of SGMAN would have correctly collected state information on earlier crash events.

5.8 Operating Voltage

The data discussed in the sections above were taken using low voltage for both the core and I/O of the UT699 (2.3 and 3.0V, respectively). Data were also taken at nominal voltage and a slightly unique voltage setting (I/O voltage set at 3.4V). Table 5.8-1 lists these data. Comparison of the data taken at LET = 13.2 MeV-cm^2/mg shows a dependence of about a factor of 4 in the cross section, but the statistical significance of this data is weak.

Table 5.8-1. The dependence of corrected and uncorrected errors on test conditions (core and I/O voltage).

Conditions Core, I/O	Program	LET	Flux	Fluence/ Run	Total Fluence	# Runs	# Corrected Errors	# Uncontained Errors
2.5V, 3.3V	SGMAN	8.7	1E5	3E6	1.2E7	4	0	0
2.5V, 3.3V	SGMAN	13.2	1E5	3E6	1.2E7	4	1	0
2.5V, 3.0V	SGMAN	13.2	1E5	3E6	1.8E7	6	4	0
2.3V, 3.0V	SGMAN	13.2	1E5	3E6	6E7	10	5	3
2.3V, 3.0V	SGMAN	13.2	1E5	3E6	6E7	10	12	5
2.3V, 3.0V	SGMAN	13.2	3.2E5	1.5E6	1.5E7	10	4	2
2.5V, 3.4V	SGMAN	13.2	1E5	6E6	6E7	10	9	5
2.5V, 3.4V	SGMAN	8.7	1E5	6E6	6E7	10	1	1
2.3V, 3.0V	HERT_SW	13.2	1E5	6E6	6E7	10	0	4
2.3V, 3.0V	SGMAN	21	1E5	6E6	2.4E7	4	22	8
2.3V, 3.0V	HERT_SW	21	1E5	6E6	3.6E7	6	24	3

6.0 MAESTRO DEVELOPMENT

The work on Maestro breaks down into two efforts: (1) general review efforts involving participation in collaborative developments and review meetings, and (2) specific review of materials for the purposes of documenting recommended efforts for development of Maestro testing.

6.1 General Review Efforts

During FY2010, JPL participated in remote reviews of the Maestro effort conducted at Boeing's facilities in Seattle, Washington. These efforts included detailed discussion of plans for test development, overview of the Maestro architecture, and identification of key collaborators on various items. These efforts proved invaluable for identifying the best way to move forward on developments, and to keep the NASA team prepared for the development of the program.

6.2 Triage of BTK Codes

This section provides a review of codes in the Tilera BTK for potential migration for use as SEE test codes. It also provides recommendations on the first 10 test codes that should be written. These recommendations are based on making the collected test data orthogonal to aid in analysis, providing testing of areas of the highest priority, and performing those tests that give the highest chances for successful testing.

Some significant risks are involved in the chosen codes and the test system architecture, which cannot be verified against actual hardware at this time. These are probably not significant except for one risk—when the testing starts, the chip is overwhelmed by crashes and the test system can do nothing but indicate a crash and go through a lengthy reset sequence (probably less than a few seconds, but contributing significant dead time). This risk results from the test system not currently being designed to catch L1 data cache parity exceptions. It is unclear if anything should be done to alleviate this risk.

The list of codes provided targets for the first test effort for Maestro. It should be expanded and revised for followup testing. In particular, improvements to the codes should be made as indicated Section 8.2.

6.2.1 Recommendations

1. The following functional test codes are recommended for migration to SEE codes:

   Code 1: itc_chip_bringup_test

   This is a control test. This test must be modified to lengthen the time before the boot test code returns in order to provide soak time. This test must also be modified to provide the same use of fundamental resources needed by other codes (i.e., if other codes implement exception handlers, this must as well; if other codes implement test state controls, such as loop control, then this code must as well).

   Code 2: chip_bringup_regfile_test - static

   This code targets static upset sensitivity for the register file. The current chip_bringup_regfile_test would be a template, but this would require minor modifications to enable setup of the 56 registers in the existing test, then soak radiation, then read out the 56 registers and record errors. This should be modified to include the FPU registers that are used in Code 6 (see chip_bringup_fpu_reg_test.c for guidance, but it is a c file).

Code 3: chip_bringup_regfile_test – dynamic

Code 3 is the same as code 2 except that it is constantly reading and writing the values in the registers. This helps identify differences between static and dynamic use of the registers. This code could be based on code 2 with minor modifications.

Code 4: L1 cache test

This code is a merged code for testing the L1 data cache. (There is no apparent test code for checking the L1 instruction cache.) This code should use the information in itc_chip_bringup_l1_data_test to form the basis of the test. If there is no time, this test can be used as the entire L1 cache test, however it is recommended to merge the itc_chip_bringup_l1_tag_test in order to enable identification of tag errors separately from data errors (note that the current test system would be unable to differentiate the different reported errors, so the benefit from merging is limited at this time). (itc_bringup_l1_dapaths_test should not be merged here since it is designed for catching weak or failed bits and has very low duty cycle.)

Code 5: L2 cache test

This code is very similar to code 4. There are also itc_chip_bringup_l2_data_test and itc_chip_bringup_l2_tag_test files that are similar to those for code 4. In migrating, all the issues with code 4 still hold. The main test to modify to fit our needs is the data test. The tag test would be useful for identifying tag errors, but is not necessary since its detailed findings will be lost until the reporting structure of the test system is modified.

Code 6: itc_chip_bringup_fpu_test

The level of priority for this code is the same as code 7 below. They should be done in order of which one is easiest. The FPU test already performs a lot of calculations. It only needs to be modified to have it perform these tests for a prolonged period of time.

Code 7: itc_imesh_test

The mesh networks basic operation should be tested as much as possible. However, this test is already written, and targets the idn and udn. The idn is the highest risk (based on a breakdown of weakest elements in Maestro). This test is currently very limited and has very little throughput. It will probably require some serious development to make it hit the idn with high duty cycle, and it is unclear exactly how to extend the operating time without good understanding of delay cycles in the test.

Code 8: itc_mem_test

This is the last code based on existing ITC-specific code development. This code is used to check the ability of the tiles to store and retrieve data from external memory. It is unclear how many or which of the DDR controllers are used here. It will be difficult to make this code more suited for SEE testing until the actual hardware is available.

Code 9: chip_bringup_xaui_test

(Warning, this is based on a code build architecture for which we have no demonstration hardware and software.) The weakest part of the Maestro after the tiles themselves is the CRC portion of the XGBE system. To begin testing that system we need to have a viable XAUI setup. This test begins to address that need. It is a loopback test that is written in c. It can be easily modified to run longer. This code also provides more test data for the idn as the XAUI is controlled over the idn.

Code 10: Reserved for later

This should be an MSHIM test, but we have no insight into the operation of the shims test to determine if it would be suitable for this test.

2. Due to the complex nature of the ITC device and the inherent expected sensitivity of the L1 data cache, it is not recommended to run complex systems until understanding of simple substructure radiation sensitivity is obtained.
    - There is a risk that upon going to the test facility, severe SEE sensitivity is observed that renders even simple codes non-operational due to lack of visibility in the test system (i.e., L1 data cache problems may dominate our results.)
    - In the event that such sensitivity is seen, having a very simple set of codes that can be set to operate for very limited amounts of time provides a viable trouble-shooting tool for identifying the source of sensitivity in order to address it for future testing of more complex test codes.
    - It may be the case that the test bootup and standard operation support should handle L1 data cache parity exceptions and report to the master tile that the test code failed, giving a code indicating parity exception. (The current test system would be unable to report this, but it at least would not waste test time with reset or other operations to recover operation of a crashed tile.)

3. Test codes should be modified to make them run for at least 90% of the time. This means the the load and execute time for one cycle of the test code needs to be known. Then, each code must be modified to either loop its test, or its soak time must be increased so that test code execution meets this 90%.

4. Comprehensive basic test system architecture should be followed.
    - The test system needs a common software structure so that results from different tests are due to the different operating blocks in the processor.
        - Currently, the test system has a common interface. The interface provides standard configuration of the tiles and transport of boot code. As indicated in recommendation 1, code 1, however, the itc_chip_bringup_test will have to be modified to provide whatever resources other codes require to operate.

    - The structure is picked up by the "#include" and link clauses that pull in generic boot codes. In particular, most of the ITC test codes use "itc_chip_bringup_l1boot.S" or "itc_chip_bringup_l1boot_pwr.S." The _pwr codes provide insight into Boeing's plan for developing SEE test code. Should NASA choose to use it, they may be a good starting point for examination.
- The software items must be expanded in operating time. Currently, they only do one quick sampling of the operation of a particular test.
    - Most of the test codes perform a basic test. This test could simply be looped N times in order to reach hundreds of milliseconds or longer.
- Most of the test codes are written around sending single packets or writing and reading one time to a location or handful of locations. This leaves the test system spending the majority of its time executing code other than the code targeting the desired subsystem. This must be modified.
    - Of particular importance will be those codes that wait on the operation of other subsystems, such as message routing codes. If a tile spends most of its time polling for receiving a message, rather than actually sending and receiving messages, then the duty cycle of that test is severely reduced.

7.0 REVIEW OF FINDINGS

This section summarizes key findings from this year's work. The key findings for this year are (1) SEEs that cause anomalies on SOCs may be difficult to isolate, but that isolation may be very important, particularly for RHBD devices, (2) interaction with manufacturers helps avoid significant missteps in identification of vulnerabilities in devices, and (3) fault isolation in complex devices is very important for space missions, but may not be supported by SOC devices.

7.1 UT699 Results

The UT699 was found to a have low cross section for sensitivity to an SEE that can cause erroneous operation or data corruption. This event was found to have a threshold of about LET = 8 MeV-cm^2/mg and a saturated cross section of about 1e-6cm^2 per device. The remainder of the test data was shown to be consistent with earlier data. In fact, all other upset modes in the UT699 are likely benign due to the built-in fault tolerance which can correct the upsets.

The event discussed above was tested for flux and fluence dependence. Flux dependence arguments hold that what matters is the probability that an upset occurs while the fault tolerance system is still recovering from another upset. This sort of interdependence carries a probability that increases linearly with the flux. Thus, if we change the flux by a factor of 33 (up or down) we expect the observed cross section to change by a factor of 33 (in the same direction). This was not seen, and in fact was in the wrong direction. Although the statistical significance is not overly compelling, no statistical argument can be made in support of flux dependence given the collected data. The same argument holds true for fluence dependence.

It is possible that the remaining upset modes observed in the UT699 (including program counter and UART configuration bit changes) may be due to single SEU sensitivity; however, all of the data collected here indicate that any sensitivity like this is below a cross section of 1e-7cm^2 and has a threshold of at least LET = 9 MeV-cm^2/mg.

7.2 Maestro Results

For this work, there are no direct results to review for Maestro. However, this report does detail the recommended path for modifying software in the BTK for use as test software for Maestro. This testing is under current development and will be performed on this NEPP task during FY2011.

8.0 FUTURE WORK

This effort is part of an ongoing task to determine test methods for SOCs that are the most effective for evaluating SEE sensitivity of devices and that determine those sensitivities of the highest interest to NASA. In order to continue this development, there are three main lines of effort recommended for next year. The first is to determine a follow up SOC device, which can be used to apply the techniques learned thus far. The second is to continue development of Maestro radiation test knowledge base and support test efforts. The final line of effort is to review test methods with particular emphasis on identifying those areas where test methods may be made to fit NASA needs more closely.

8.1 Next SOC for Study

There are several options for selection of an SOC for continuation of this work. The most likely candidates require additional study to determine the most appropriate one. Table 8.1-1 lists the initial suggestions for viewing.

Table 8.1-1. Potential Candidates for Further SOC Testing

Device
Atmel AT913, LEON 2FT, close competitor to UT699
Freescale MPC8640 Dual-Core e600
SiLabs C8051F310 Microcontroller

8.2 Upcoming Maestro Efforts

Maestro SEE testing is now primarily being handled by NASA. JPL will participate in this testing in the development stages and in the test operations stages. JPL's role at this stage is to review the software for test and to determine its applicability for radiation testing. JPL will continue to advise on general software structure to ensure that test efforts have the highest chances for success.

The major thrusts for this topic are to look at the overall Maestro operations code to ensure that the architecture of a running system is consistent with the types of upsets that may occur during testing. Of particular interest will be making certain that unexpected event types will not cause unnecessary limitations on collected data sets.

8.3 Community Engagement

Efforts have been carried out under this task to identify a support network to assist in appropriate testing for Maestro. This has been a difficult task and will likely only get better with additional efforts to coordinate. Coordination between JPL's advanced computer architecture group and the radiation effects group will continue to be sought to improve the coverage of key issues related to fault isolation for Maestro.

Additionally, a limited set of microprocessor developers at JPL has been identified. If the task permits, that group will be developed into a resource for identifying general SOC and microprocessor radiation test methods for greatest benefit to hardware and software developers. This effort is only in its infancy.

9.0 REFERENCES

[1] Aeroflex. *UT699 LEON 3FT/Sparc™ V8 Microprocessor Advanced Users Manual*. September 2009.

[2] Hafer, C., et al., “LEON 3FT Processor Radiation Effects Data,” *Radiation Effects Data Workshop*, IEEE, 2009.

APPENDIX A: ACRONYMS

AMFRT	Architecture for Microprocessor Functional Radiation Testing
BTK	Board Test Kit
CCR	Cache Condition Register
DUT	device under test
EDAC	error detection and correction
FT	fault tolerant
FY	fiscal year
GPR	General Purpose Register
I/O	input/output
JPL	Jet Propulsion Laboratory
LET	linear energy transfer
MMU	memory management unit
NASA	National Aeronautics and Space Administration
NEPP	NASA Electronic Parts and Packaging Program
RHBD	radiation hardened by design
SEE	single-event effects
SOC	system on a chip
TAMU	Texas A&M University Cyclotron
UART	Universal Asynchronous Receiver/Transmitter